



U.S. Department of Energy



Office of Science  
Department of Energy

# Using the TAU Performance Analysis System on BG/P INCITE Performance Workshop

ALCF INCITE Workshop Series

Argonne National Laboratory

May 7<sup>th</sup> & 8<sup>th</sup>, 2008

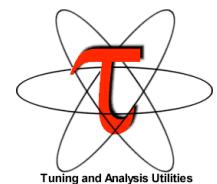
Kevin Huck

[khuck@cs.uoregon.edu](mailto:khuck@cs.uoregon.edu)

University of Oregon



UNIVERSITY  
OF OREGON



# Outline

---

- Introduction to performance evaluation
- TAU introduction
- Hands-on tutorial for using TAU on BG/P at Argonne

# Performance Evaluation

---

- Profiling
  - Presents summary statistics of performance metrics
    - number of times a routine was invoked
    - exclusive, inclusive time/hpm counts spent executing it
    - number of instrumented child routines invoked, etc.
    - structure of invocations (calltrees/callgraphs)
    - memory, message communication sizes also tracked
- Tracing
  - Presents when and where events took place along a global timeline
  - timestamped log of events
  - message communication events (sends/receives) are tracked
    - shows when and where messages were sent
  - large volume of performance data generated leads to more perturbation in the program

# Definitions – Profiling

---

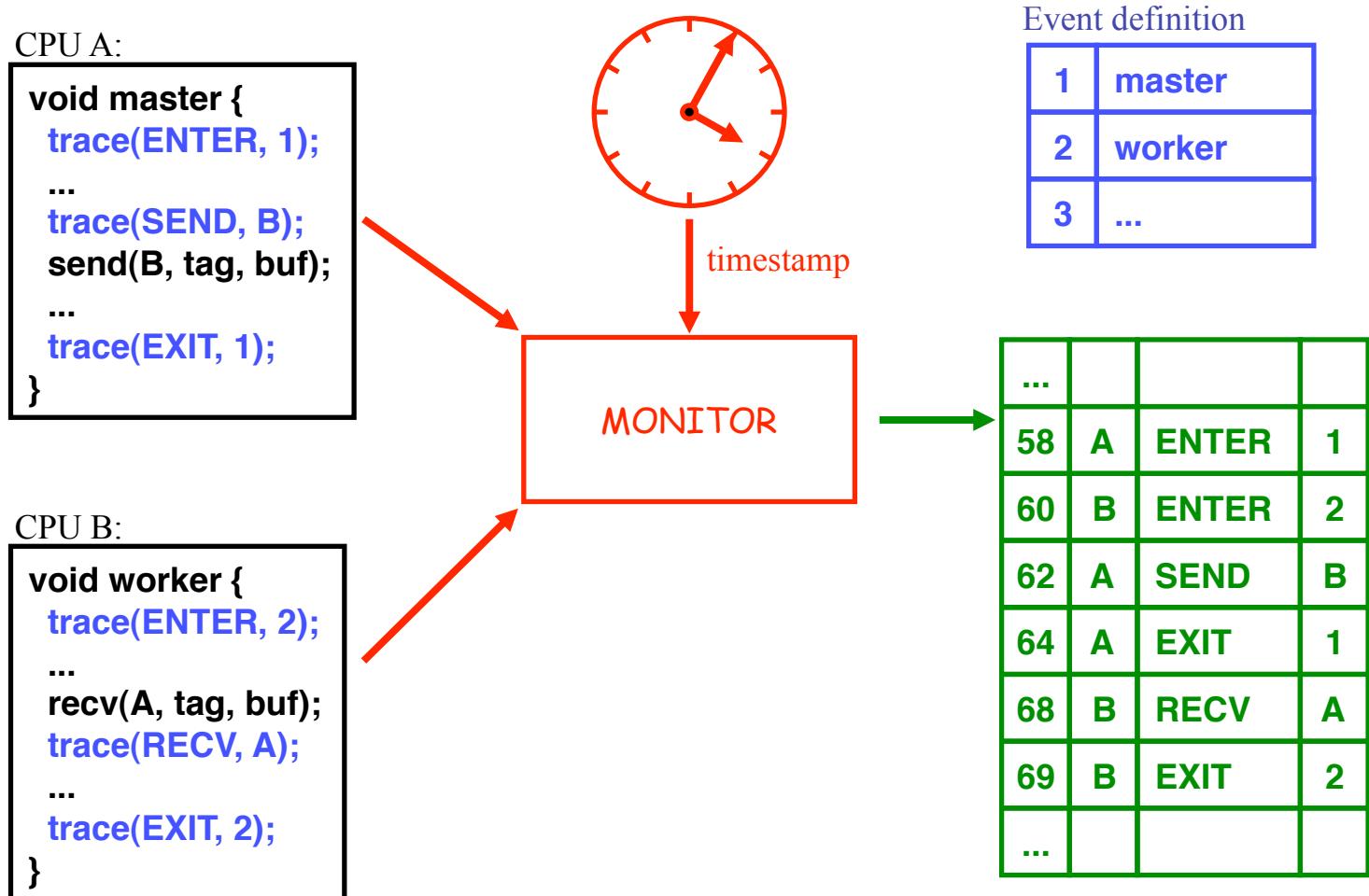
- Profiling
  - Recording of summary information during execution
    - inclusive, exclusive time, # calls, hardware statistics, ...
  - Reflects performance behavior of program entities
    - functions, loops, basic blocks
    - user-defined “semantic” entities
  - Very good for low-cost performance assessment
  - Helps to expose performance bottlenecks and hotspots
  - Implemented through
    - sampling: periodic OS interrupts or hardware counter traps
    - instrumentation: direct insertion of measurement code

# Definitions – Tracing

---

- **Tracing**
  - Recording of information about significant points (**events**) during program execution
    - entering/exiting code region (function, loop, block, ...)
    - thread/process interactions (e.g., send/receive message)
  - Save information in **event record**
    - timestamp
    - CPU identifier, thread identifier
    - Event type and event-specific information
  - **Event trace** is a time-sequenced stream of event records
  - Can be used to reconstruct dynamic program behavior
  - Typically requires code instrumentation

# Event Tracing: Instrumentation, Monitor, Trace

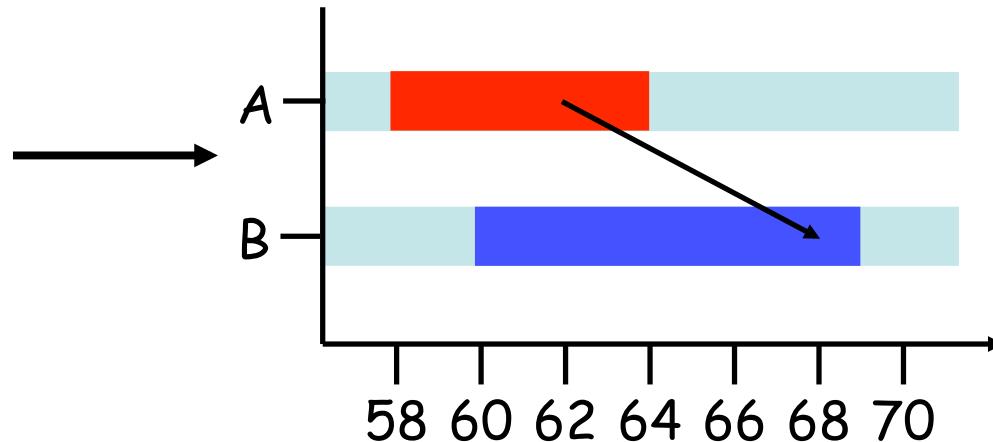


# Event Tracing: “Timeline” Visualization

1	master
2	worker
3	...



...				
58	A	ENTER	1	
60	B	ENTER	2	
62	A	SEND	B	
64	A	EXIT	1	
68	B	RECV	A	
69	B	EXIT	2	
...				



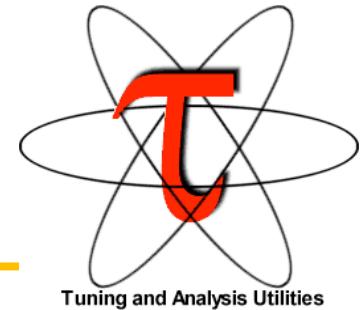
# Steps of Performance Evaluation

---

- Collect basic routine-level timing profile to determine where most time is being spent
- Collect routine-level hardware counter data to determine types of performance problems
- Collect callpath profiles to determine sequence of events causing performance problems
- Conduct finer-grained profiling and/or tracing to pinpoint performance bottlenecks
  - Loop-level profiling with hardware counters
  - Tracing of communication operations

# TAU Parallel Performance System

---



- <http://tau.uoregon.edu/>
- Multi-level performance instrumentation
  - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
  - Computer system architectures and operating systems
  - Different programming languages and compilers
- Support for multiple parallel programming paradigms
  - Multi-threading, message passing, mixed-mode, hybrid
- Integration in complex software, systems, applications

# Using TAU: A brief Introduction

---

- To instrument source code:

```
% setenv TAU_MAKEFILE /usr/apps/tools/tau/tau-2.17.1/  
x86_64/lib/Makefile.tau-mpi-pdt-pgi
```

And use tau\_f90.sh, tau\_cxx.sh or tau\_cc.sh as Fortran, C++ or C compilers:

```
% mpif90 foo.f90
```

changes to

```
% tau_f90.sh foo.f90
```

- Execute application and then run:

```
% pprof (for text based profile display)
```

```
% paraprof (for GUI)
```

# Performance Tools FAQ/Concerns

---

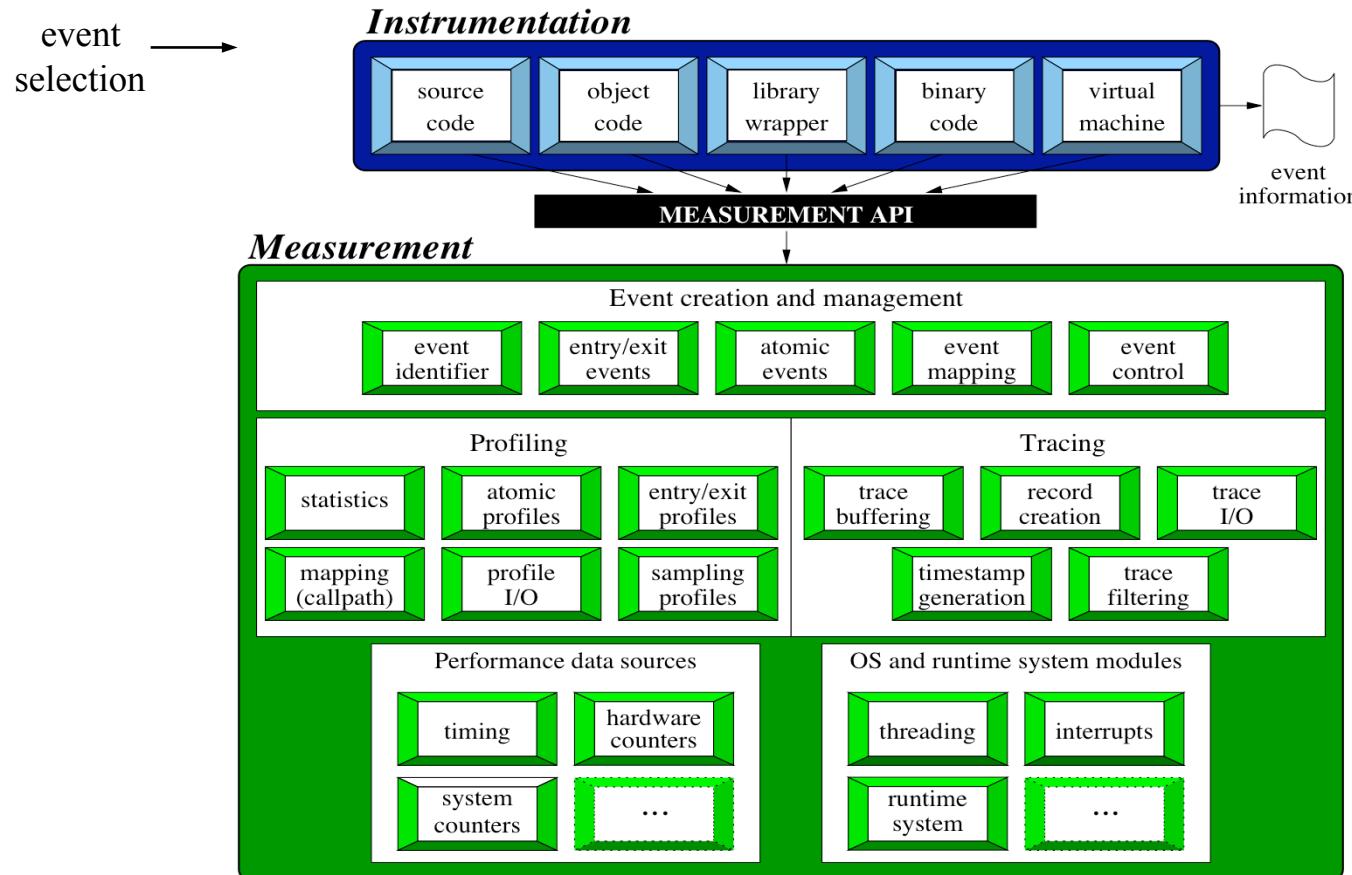
- Does it automatically instrument my code? At the routine level? At the outer-loop level?
- Can it show me where time is spent in my code? PAPI Flops? L1 data cache misses? Can I measure more than one quantity in a trial?
- Does the tool support profiling (runtime summarization) as well as tracing (time-line based displays)? What about profile snapshots? Callpath (parent-child) profiles? Can I use it to easily benchmark codes?
- Can I observe the performance data at runtime as the application executes?
- Can it show me memory utilization? Memory leaks? Mallocs/frees? When and where?
- What about I/O? Can I observe bandwidth of reads/writes? Volume of I/O? What about Kernel events? User space+Kernel?
- What is the typical overhead? Can I reduce it to < 5%? < 1%? Can it compensate and remove timer overhead from performance data? Can it throttle away instrumentation in lightweight routines at runtime to reduce overhead?
- I already have profile data from <XYZ> tool. Can it import my legacy data?
- I prefer <XYZ> performance tool for visualization. Can it hook up with this tool? Are there converters?

# Performance Tools FAQ/Concerns (contd.)

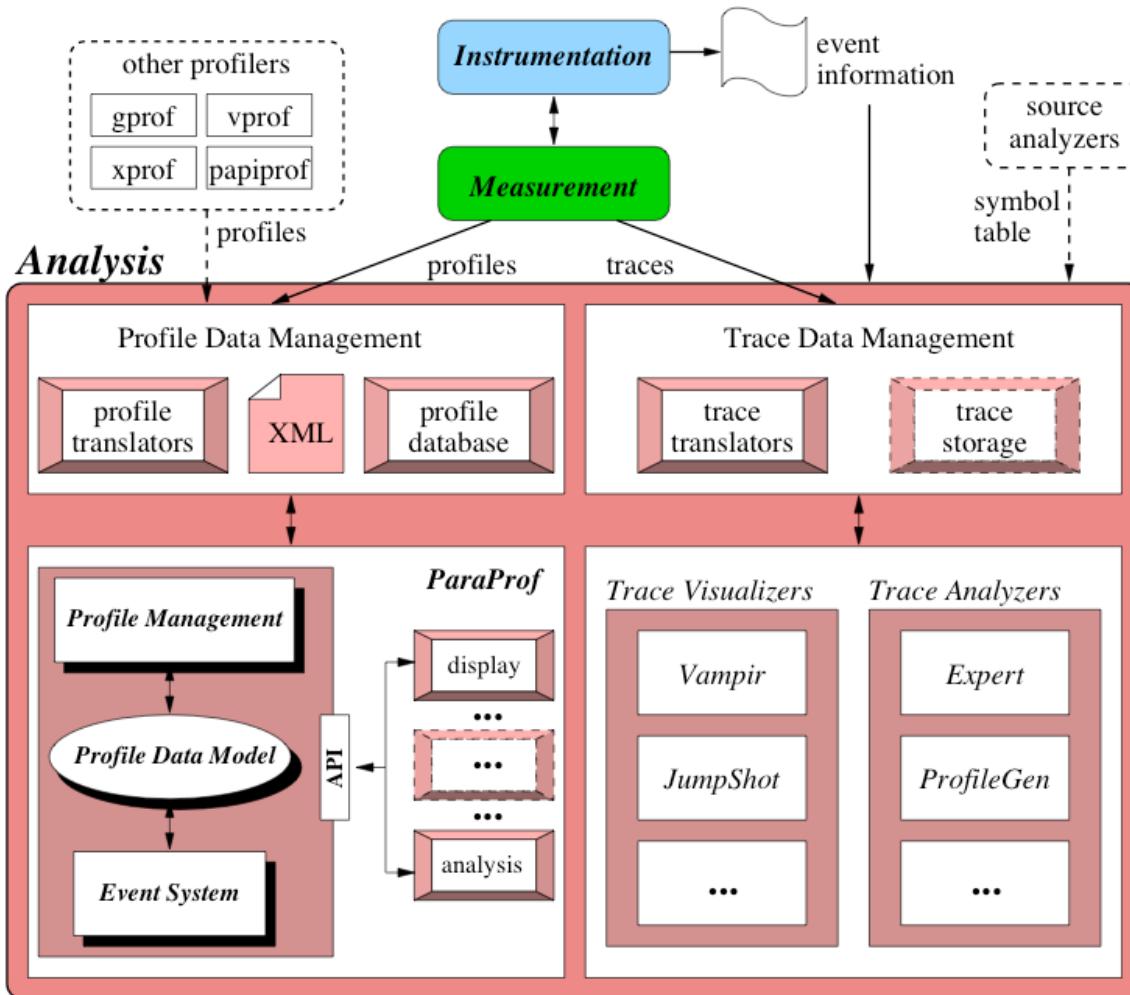
---

- Can I use it for multi-core CPUs? Compare the performance of application running on a single vs. multi-core processor? Can I observe multi-core data snoops, invalidates?
- Can I share the performance data with my colleagues in a secure manner (web/database)? Can it automatically track progress of my application over time (~ 6 mos)? Can I use it for scalability studies? Over multiple platforms?
- Are the GUI client tools available under Linux? MS Windows? Apple?
- Does it run on all Cray, IBM, SGI, HP ... platforms? CNL? Catamount?
- Does it support MPI? MPI2? Threads? Hybrid MPI+Pthreads/MPI+OpenMP?
- Does it support Fortran? C++, C? Java? Python? Python+MPI+F90+C++...?
- Does it support Intel/PGI/PathScale/IBM/Cray/Sun compilers?
- Are tools available in command-line form & GUI? IDE GUI? Web-based? 3D?
- Is it already installed and supported on my HPC system? What about systems at NERSC? ANL? LLNL? LANL? NASA? DoD? NSF sites?...
- Is there support (phone/e-mail) available for the tool? Professional support? For instrumentation? Analysis?
- Will it work on the new <XYZ> HPC platform scheduled for release six months from now?
- Is it free? BSD license? ...

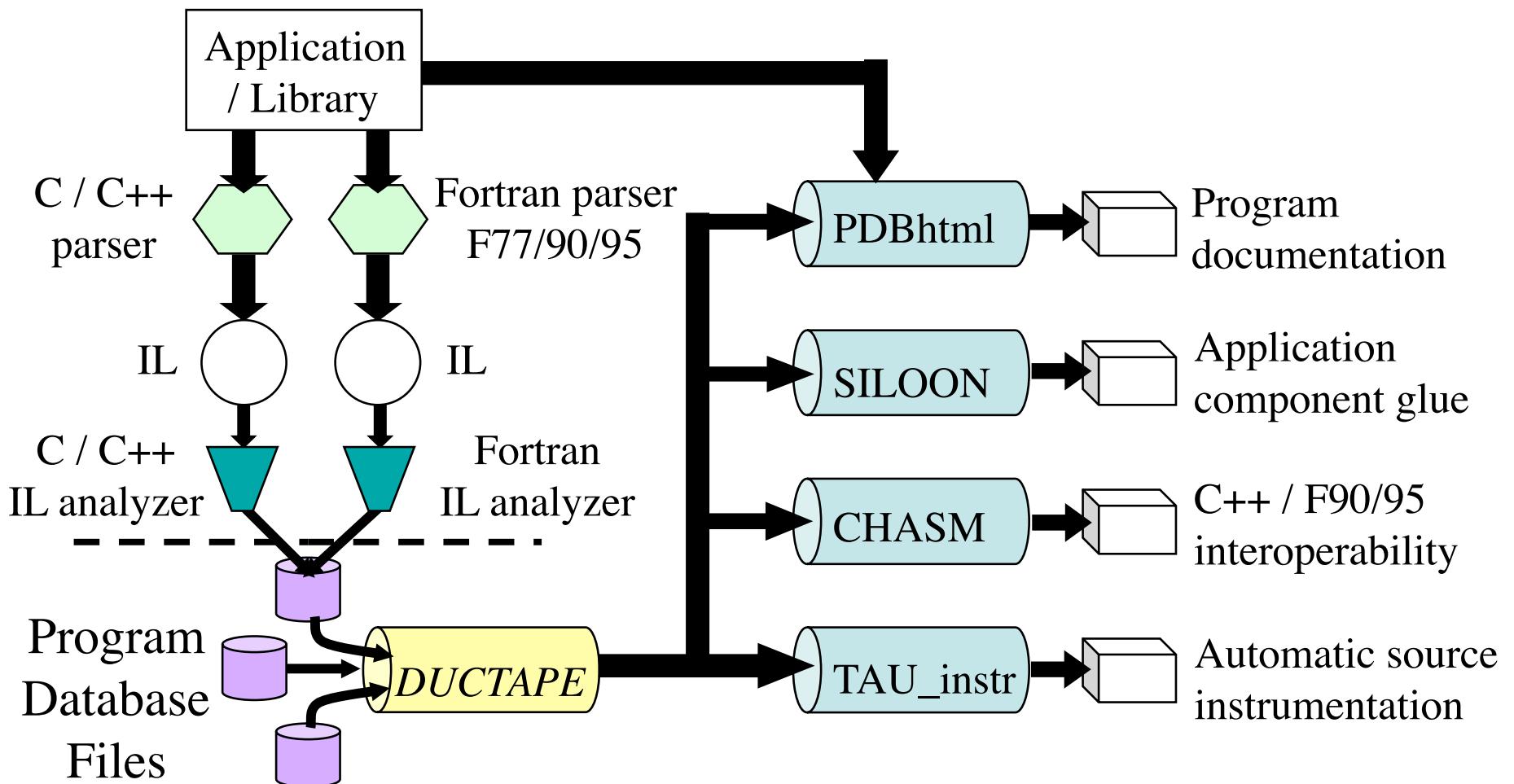
# TAU Performance System Architecture



# TAU Performance System Architecture



# Program Database Toolkit (PDT)



# TAU Instrumentation Approach

---

- Support for standard program events
  - Routines
  - Classes and templates
  - Statement-level blocks
- Support for user-defined events
  - Begin/End events (“user-defined timers”)
  - Atomic events (e.g., size of memory allocated/freed)
  - Selection of event statistics
- Support definition of “semantic” entities for mapping
- Support for event groups
- Instrumentation optimization (eliminate instrumentation in lightweight routines)

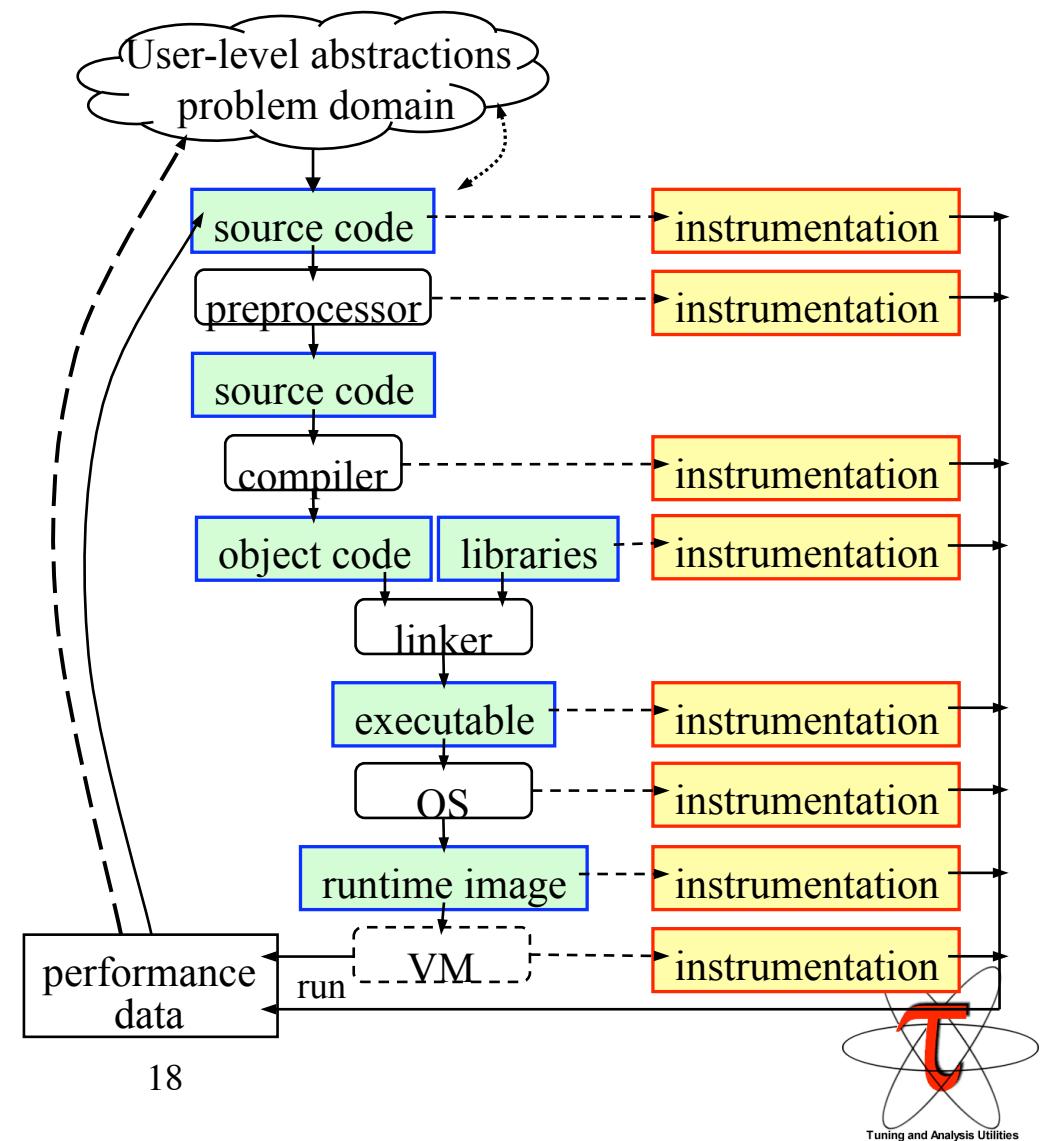
# TAU Instrumentation

---

- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual (TAU API, TAU Component API)
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (*PDT*))
      - OpenMP (directive rewriting (*Opari*), *POMP spec*)
  - Object code
    - pre-instrumented libraries (e.g., MPI using *PMPI*)
    - statically-linked and dynamically-linked
  - Executable code
    - dynamic instrumentation (pre-execution) (*DynInstAPI*)
    - virtual machine instrumentation (e.g., Java using *JVMPPI*)
    - Python interpreter based instrumentation at runtime
  - Proxy Components

# Multi-Level Instrumentation and Mapping

- Multiple instrumentation interfaces
- Information sharing
  - Between interfaces
- Event selection
  - Within/between levels
- Mapping
  - Associate performance data with high-level semantic abstractions
- Instrumentation targets measurement API with support for mapping



# TAU Measurement Approach

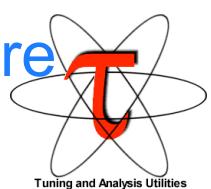
---

- Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
  - Online performance profile overhead compensation
- Portable and scalable parallel tracing solution
  - Trace translation to Open Trace Format (OTF)
  - Trace streams and hierarchical trace merging
- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)



UNIVERSITY  
OF OREGON

Performance measurement for CCA component software



# Using TAU

---

- Configuration
- Instrumentation
  - Manual
  - MPI – Wrapper interposition library
  - PDT- Source rewriting for C,C++, F77/90/95
  - OpenMP – Directive rewriting
  - Component based instrumentation – Proxy components
  - Binary Instrumentation
    - DyninstAPI – Runtime Instrumentation/Rewriting binary
    - Java – Runtime instrumentation
    - Python – Runtime instrumentation
- Measurement
- Performance Analysis

# TAU Measurement System Configuration

---

- **configure [OPTIONS]**

```
{-c++=<CC>, -cc=<cc>}  
-pdt=<dir>  
-opari=<dir>  
-papi=<dir>  
-vampirtrace=<dir>  
-mpi[inc/lib]=<dir>  
-dyninst=<dir>  
-shmem[inc/lib]=<dir>  
-python[inc/lib]=<dir>  
-tag=<name>  
-epilog=<dir>  
-slog2  
-otf=<dir>  
-arch=<architecture>  
  
{-pthread, -sproc}  
-openmp  
-jdk=<dir>  
-fortran=[vendor]
```

Specify C++ and C compilers  
Specify location of PDT  
Specify location of O pari OpenMP tool  
Specify location of PAPI  
Specify location of VampirTrace  
Specify MPI library instrumentation  
Specify location of DynInst Package  
Specify PSHMEM library instrumentation  
Specify Python instrumentation  
Specify a unique configuration name  
Specify location of EPILOG  
Build SLOG2/Jumpshot tracing package  
Specify location of OTF trace package  
Specify architecture explicitly  
    (bgl, xt3, ibm64, ibm64linux...)  
Use pthread or SGI sproc threads  
Use OpenMP threads  
Specify Java instrumentation (JDK)  
Specify Fortran compiler



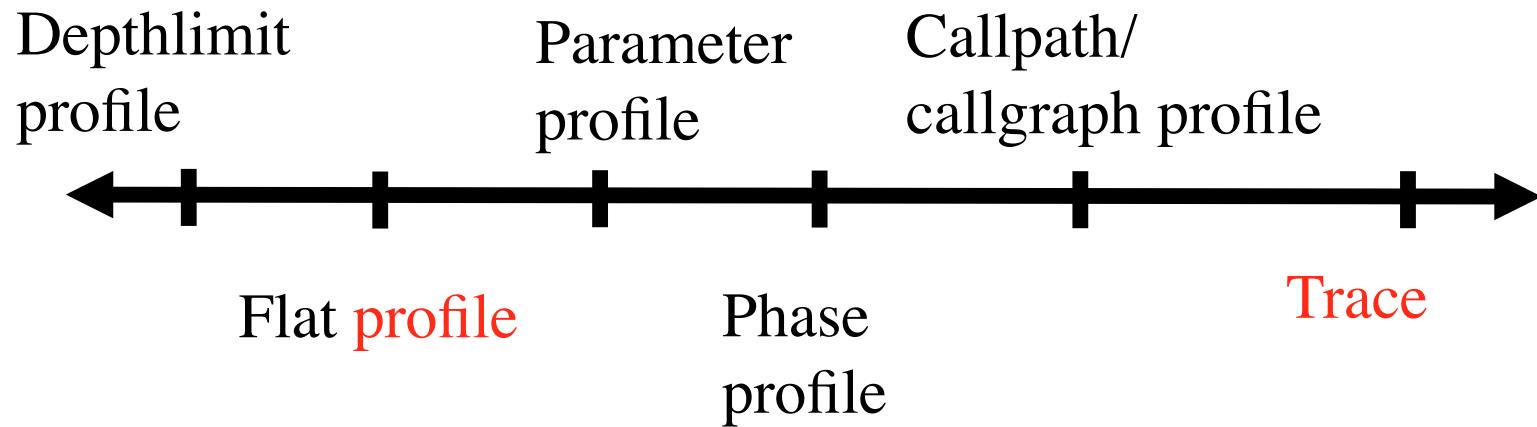
# TAU Measurement System Configuration

---

- configure [OPTIONS]
  - TRACE Generate binary TAU traces
  - PROFILE (default) Generate profiles (summary)
  - PROFILECALLPATH Generate call path profiles
  - PROFILEPHASE Generate phase based profiles
  - PROFILEMEMORY Track heap memory for each routine
  - PROFILEHEADROOM Track memory headroom to grow
  - MULTIPLECOUNTERS Use hardware counters + time
  - COMPENSATE Compensate timer overhead
  - CPUTIME Use usertime+system time
  - PAPIWALLCLOCK Use PAPI's wallclock time
  - PAPIVIRTUAL Use PAPI's process virtual time
  - SGITIMERS Use fast IRIX timers
  - LINUXTIMERS Use fast x86 Linux timers

# Performance Evaluation Alternatives

---



Each alternative has:

- one metric/counter
- multiple counters

Volume of performance data

# TAU's MPI Wrapper Interposition Library

---

- Uses standard MPI Profiling Interface
  - Provides name shifted interface
    - MPI\_Send = PMPI\_Send
    - Weak bindings
- Interpose TAU's MPI wrapper library between MPI and TAU
  - `-lmpi` replaced by `-lTauMpi -lpmmpi -lmpi`
- No change to the source code! Just **re-link** the application to generate performance data
  - `setenv TAU_MAKEFILE <dir>/<arch>/lib/Makefile.tau-mpi-[options]`
  - Use `tau_cxx.sh`, `tau_f90.sh` and `tau_cc.sh` as compilers

# Auto Instrumentation using TAU\_COMPILER

---

- `$(TAU_COMPILER)` stub Makefile variable
- Invokes PDT parser, TAU instrumentor, compiler through `tau_compiler.sh` shell script
- Requires minimal changes to application Makefile
  - Compilation rules are not changed
  - User adds `$(TAU_COMPILER)` before compiler name
    - F90=mpxlf90  
Changes to  
F90= `$(TAU_COMPILER)` mpxlf90
- Passes options from TAU stub Makefile to the four compilation stages
- Use `tau_cxx.sh`, `tau_cc.sh`, `tau_f90.sh` scripts **OR** `$(TAU_COMPILER)`
- Uses original compilation command if an error occurs

# TAU Manual Instrumentation API for C/C++

---

- Initialization and runtime configuration
  - `TAU_PROFILE_INIT(argc, argv);`  
`TAU_PROFILE_SET_NODE(myNode);`  
`TAU_PROFILE_SET_CONTEXT(myContext);`  
`TAU_PROFILE_EXIT(message);`  
`TAU_REGISTER_THREAD();`
- Function and class methods for C++ only:
  - `TAU_PROFILE(name, type, group);`
  - `TAU_PROFILE ( name, type, group);`
- Name-based API
  - `TAU_START("timer_name");`  
`TAU_STOP("timer_name");`
- User-defined timing
  - `TAU_PROFILE_TIMER(timer, name, type, group);`  
`TAU_PROFILE_START(timer);`  
`TAU_PROFILE_STOP(timer);`

# TAU Measurement API (continued)

---

- Defining application phases
  - TAU\_PHASE\_CREATE\_STATIC( var, name, type, group);
  - TAU\_PHASE\_CREATE\_DYNAMIC( var, name, type, group);
  - TAU\_PHASE\_START(var)
  - TAU\_PHASE\_STOP (var)
- User-defined events
  - TAU\_REGISTER\_EVENT(variable, event\_name);
  - TAU\_EVENT(variable, value);
  - TAU\_PROFILE\_STMT(statement);
- Heap Memory Tracking:
  - TAU\_TRACK\_MEMORY();
  - TAU\_TRACK\_MEMORY\_HEADROOM();
  - TAU\_SET\_INTERRUPT\_INTERVAL(seconds);
  - TAU\_DISABLE\_TRACKING\_MEMORY[ HEADROOM]();
  - TAU\_ENABLE\_TRACKING\_MEMORY[ HEADROOM]();

# Tutorial Overview

---

- Getting example code
- Setting up environment
- Building and running experiments
- Visualizing output with pprof
- Visualizing output with ParaProf
- Saving output to PerfDMF
- Analysis with PerfExplorer

# TAU installation location

---

- TAU is installed in /soft/apps/tau/tau\_latest
  - 2.17
  - 2.17.1
  - tau\_latest
  - PDT
  - tau.bashrc
  - tau.cshrc
- We will use /soft/apps/tau/tau\_latest for the demo
- Examples are in /soft/apps/tau/tau\_latest/examples

# Notes about TAU on BG/P

---

- Front end nodes are ppc64
- Back end nodes are bgp
- TAU interactive tools are built for ppc64 or Java
- Back end tools (measurement) are built for bgp
- Configurations available (various combinations):
  - TAU (with PDT)
  - MPI
  - pthreads
  - Callpath
  - Multiple counters
  - Compensate

# TAU examples

---

- /soft/apps/tau/tau\_latest/examples has many examples to demonstrate different TAU features.
- We will use the “matmult” example
  - Multiply two matrices (2000x2000)
  - MPI, F90
- We will build:
  - Without TAU
  - With TAU profiles
  - With TAU profiles and selective instrumentation
  - With TAU profiles + callpath
  - With TAU profiles + callpath + PAPI
  - With TAU tracing

# Copy matmult example locally

---

```
cp -r /soft/apps/tau/tau_latest/examples/matmult ~/.
```

# Set up your TAU environment

---

- Source /soft/apps/tau/tau.cshrc or tau.bashrc

```
# TAU and PDT
setenv TAUARCHITECTURE ppc64
setenv TAU /soft/apps/tau/tau_latest/bgp/lib

# even if this TAUVERSION is incorrect, it does not matter.
setenv TAUVERSION tau-2.17.1

# TAU
set path=($path /soft/apps/tau/pdtoolkit-3.12/$TAUARCHITECTURE/bin)
set path=(/soft/apps/tau/tau_latest/$TAUARCHITECTURE/bin $path)

# TAU counters. Choose default counters for -multiplecounter option in TAU.
setenv COUNTER1 GET_TIME_OF_DAY

# disable instrumentation in lightweight routines at runtime
setenv TAU_THROTTLE 1

# set LD library path
if ( $?LD_LIBRARY_PATH ) then
    setenv LD_LIBRARY_PATH $TAU\:$LD_LIBRARY_PATH
else
    setenv LD_LIBRARY_PATH $TAU
endif
```

# Building and running example *without* TAU

---

- mpixlf90\_r -c matmult.f90 -o matmult.o
  - mpixlf90\_r matmult.o -o matmult
- 
- qsub -t 5 -n 32 matmult

# Building and running example *with* TAU

---

- Set the TAU makefile:
  - `setenv TAU_MAKEFILE $TAU/Makefile.tau-mpi-pdt`
  - `export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt`
- Replace `mpixlf90_r` with `tau_f90.sh` (or just use `make`)
- `tau_f90.sh -c matmult.f90 -o matmult.o`
- `tau_f90.sh matmult.o -o matmult`
- `qsub -t 5 -n 32 matmult`

# TAU output

---

- There will be  $N$  profile.x.x.x files in the directory where the executable lives
- Example - profile.3.0.1:
  - Process 3 (indexed at 0)
  - Context 0 (ignore for now...)
  - Thread 1 (indexed at 0)
- For matmult with 32 processors:
  - profile.0.0.0 ... profile.31.0.0

# pprof options

---

```
usage: pprof [-c|-b|-m|-t|-e|-i|-v] [-r] [-s] [-n num] [-f filename] [-p] [-l] [-d] [node numbers]

-a : Show all location information available
-c : Sort according to number of Calls
-b : Sort according to number of subRoutines called by a function
-m : Sort according to Milliseconds (exclusive time total)
-t : Sort according to Total milliseconds (inclusive time total) (default)
-e : Sort according to Exclusive time per call (msec/call)
-i : Sort according to Inclusive time per call (total msec/call)
-v : Sort according to Standard Deviation (excl usec)
-r : Reverse sorting order
-s : print only Summary profile information
-n <num> : print only first <num> number of functions
-f filename : specify full path and Filename without node ids
-p : suppress conversion to hh:mm:ss:mmm format
-l : List all functions and exit
-d : Dump output format (for tau_reduce) [node numbers] : prints only info about all contexts/threads of given node numbers
```



# pprof output (truncated)

---

FUNCTION SUMMARY (mean):

---

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive Name
					usec/call
100.0	62	16,643	1	2318.47	16643178 MAIN
93.2	15,508	15,508	62.5	0	248128 MULTIPLY_MATRICES
3.1	507	507	125.969	0	4032 MPI_Recv()
2.3	376	376	2000	0	188 MPI_Bcast()
0.7	116	116	1	0	116762 MPI_Finalize()
0.3	50	50	1	0	50779 MPI_Init()
0.1	11	11	125.969	0	91 MPI_Send()
0.1	9	9	0.03125	0	301331 INITIALIZE
0.0	0.00803	0.00803	1	0	8 MPI_Comm_rank()
0.0	0.006	0.006	1	0	6 MPI_Comm_size()

# ParaProf Visualization

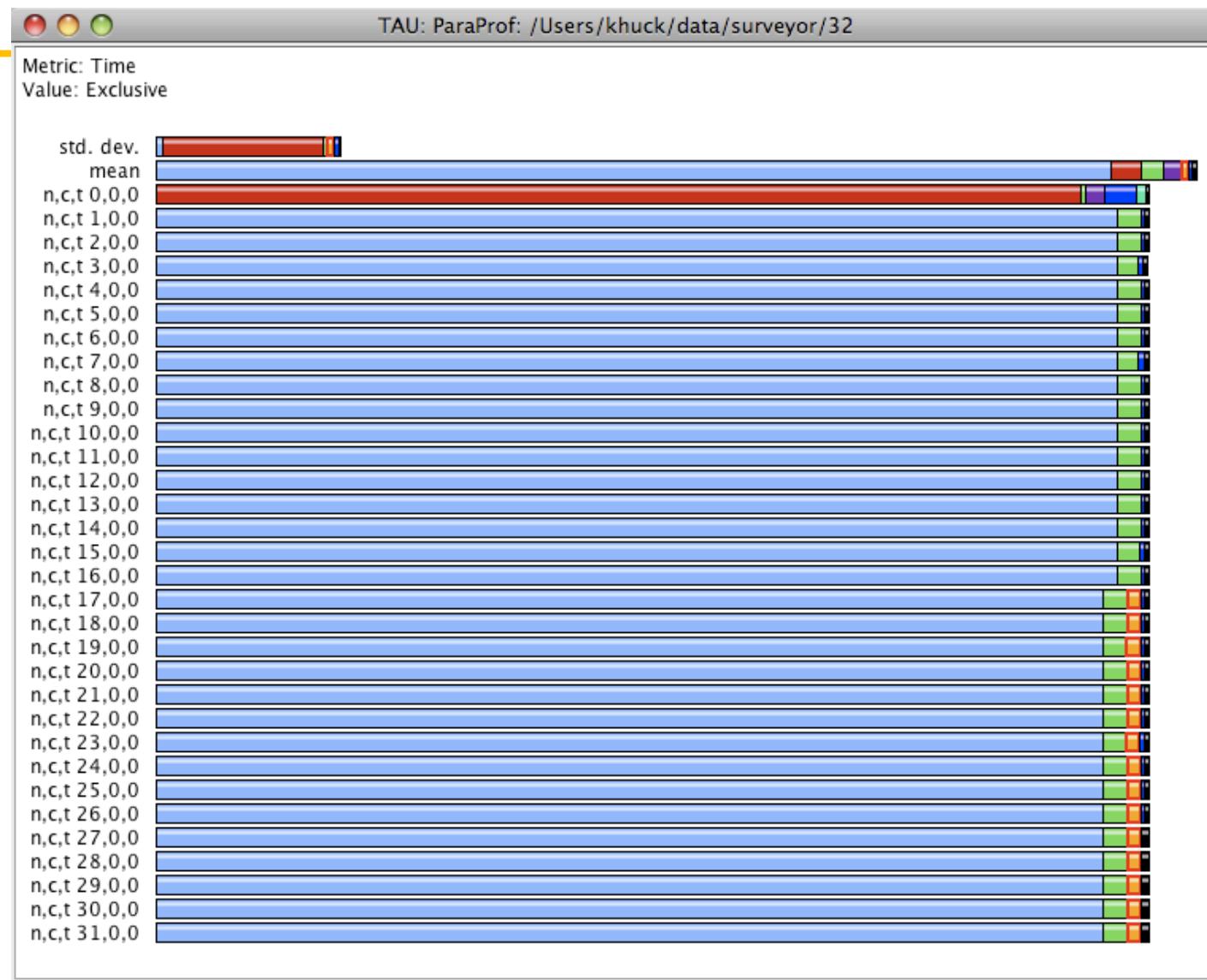
TAU: ParaProf Manager

TrialField	Value
Name	32/surveyor/data/khuck/Users/
Application ID	0
Experiment ID	0
Trial ID	0
BGP Coords	(3,3,1)
BGP DDRSize (MB)	2048
BGP Location	R00-M1-N00-J32
BGP Node Mode	Coprocessor (17821024)
BGP Processor ID	0
BGP Size	(4,4,4)
BGP isTorus	(0,0,0)
BGP numNodesInPset	1
BGP numPsets	64
BGP psetNum	0
BGP rankInPset	27
CPU Type	450 Blue Gene/P DD2
CWD	/gpfs/home/khuck/matmult/32
Executable	/sbin.rd/iproxy
Hostname	ion-9
Local Time	2008-05-05T23:24:22+00:00
MPI Processor Name	Rank 31 of 32 <3,3,1,0> R00-M1...
Memory Size	1773608 kB
Node Name	ion-9
OS Machine	BGP
OS Name	CNK
OS Release	2.6.19.2
OS Version	1
Starting Timestamp	1210029845779271
TAU Architecture	bgp
TAU Config	-arch=bgp -pdt=/soft/apps/tau/pdt...
TAU Version	2.17.1
Timestamp	1210029862578004
UTC Time	2008-05-05T23:24:22Z
pid	131

O

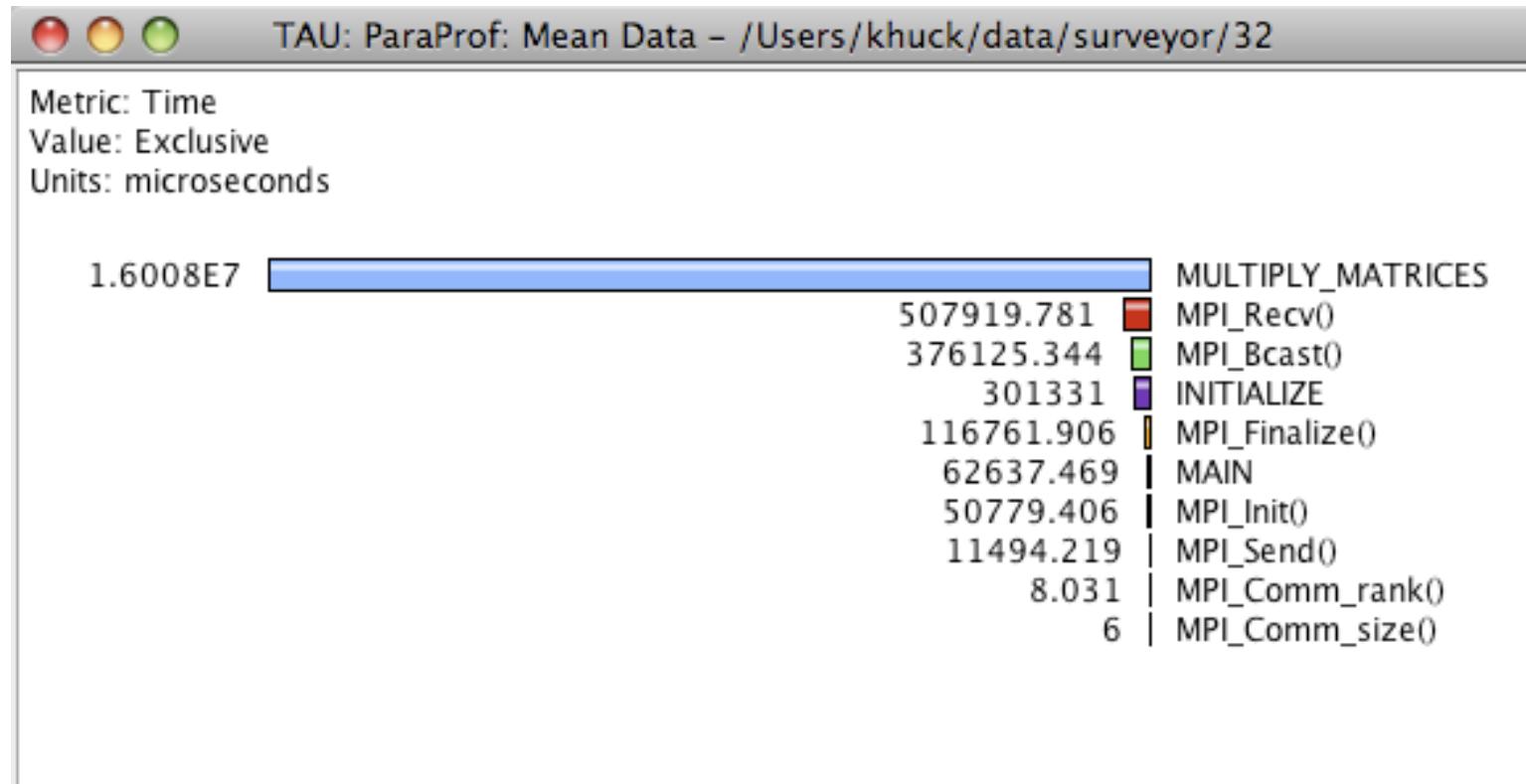
UNIVERSITY  
OF OREGON

# ParaProf visualization



40

# Mean performance



# Instrumenting loops

---

- To instrument loops, we will use selective instrumentation
- Set TAU\_OPTIONS:
  - `setenv TAU_OPTIONS '-optTauSelectFile=select.tau'`
  - `export TAU_OPTIONS='-optTauSelectFile=select.tau'`
- Rebuild (using make)
  - `make clean`
  - `make`
- Submit the job (passing environment variable)
  - `qsub -t 5 -n 32 --env TAU_THROTTLE=1 matmult`

# pprof output

---

FUNCTION SUMMARY (mean):

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive Name
					usec/call
100.0	27	16,703	1	6.0625	16703307 MAIN
93.4	2	15,606	0.96875	188.469	16110203 Loop: MAIN [{matmult.f90} {117,9}-{128,14}]
93.2	1	15,574	62.5	62.5	249187 MULTIPLY_MATRICES
93.2	15,573	15,573	62.5	0	249171 Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]
3.1	510	510	125.969	0	4049 MPI_Recv()
3.0	15	506	0.03125	125	16223292 Loop: MAIN [{matmult.f90} {86,9}-{106,14}]
2.3	12	392	0.96875	1937.5	404877 Loop: MAIN [{matmult.f90} {112,9}-{115,14}]
2.3	382	382	2000	0	191 MPI_Bcast()
0.7	117	117	1	0	117306 MPI_Finalize()
0.2	39	39	1	0	39929 MPI_Init()
0.1	11	11	125.969	0	95 MPI_Send()
0.1	0.00306	9	0.03125	0.0625	301434 INITIALIZE

...<rest deleted>

# Capturing callpath information

---

- Change the makefile:
  - `setenv TAU_MAKEFILE $TAU/Makefile.tau-callpath-mpi-pdt`
  - `export TAU_MAKEFILE=$TAU/Makefile.tau-callpath-mpi-pdt`
- Rebuild
  - `make clean`
  - `make`
- Submit the job (setting callpath depth)
  - `qsub -t 5 -n 32 --env TAU_THROTTLE=1:TAU_CALLPATH_DEPTH=500 matmult`
- Default depth is 2

# pprof output

FUNCTION SUMMARY (mean):

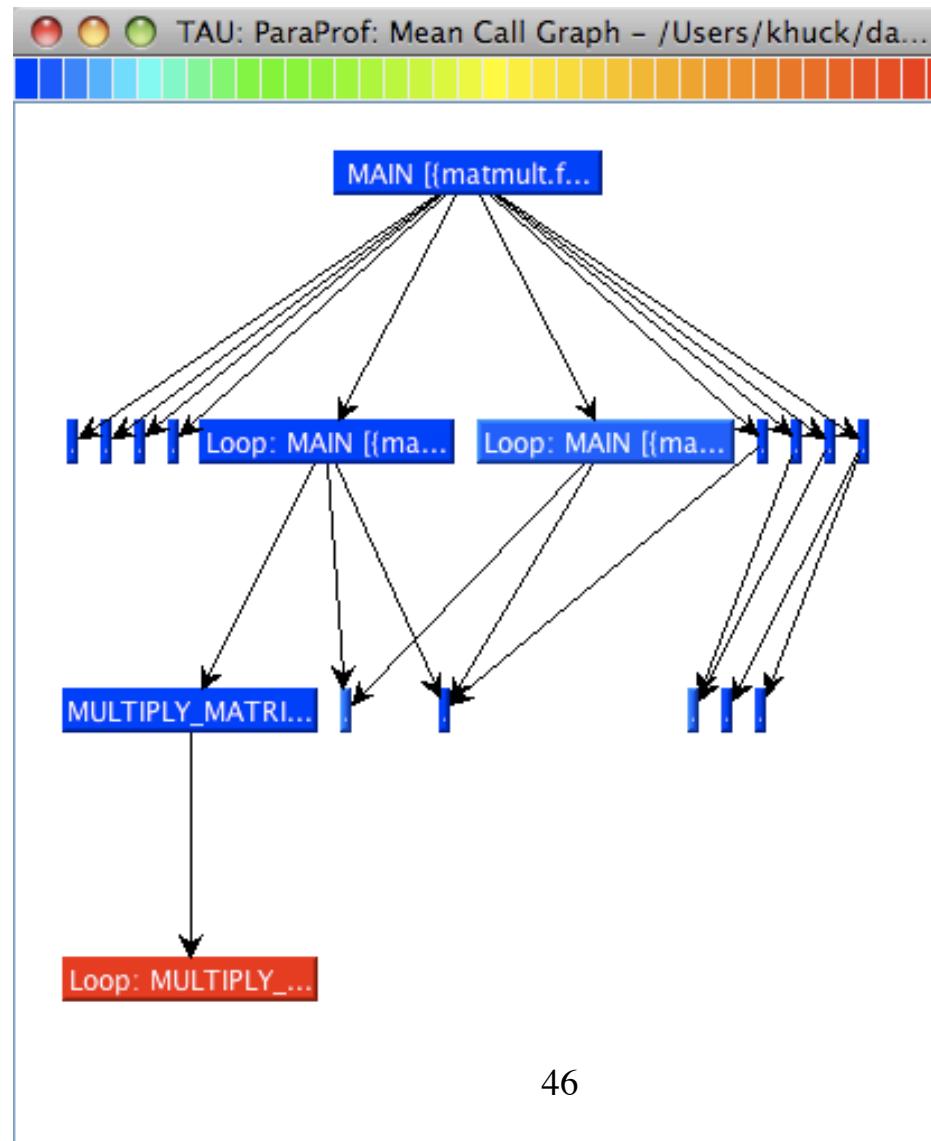
%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive Name usec/call
<hr/>					
100.0	49	16,858	1	6.0625	16858730 MAIN
92.7	2	15,620	0.96875	188.469	16124446 Loop: MAIN [{matmult.f90} {117,9}-{128,14}]
92.7	2	15,620	0.96875	188.469	16124446 MAIN [{matmult.f90} {39,15}] => Loop: MAIN [{matmult.f90} {117,9}-{128,14}]
92.4	3	15,579	62.5	62.5	249276 MAIN [{matmult.f90} {39,15}] => Loop: MAIN [{matmult.f90} {117,9}-{128,14}] => MULTIPLY_MATRICES [{matmult.f90} {25,18}]
92.4	3	15,579	62.5	62.5	249276 MULTIPLY_MATRICES
92.4	15,575	15,575	62.5	0	249215 Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]
92.4	15,575	15,575	62.5	0	249215 MAIN [{matmult.f90} {39,15}] => Loop: MAIN [{matmult.f90} {117,9}-{128,14}] => MULTIPLY_MATRICES [{matmult.f90} {25,18}] => Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]
3.0	509	509	125.969	0	4046 MPI_Recv()
3.0	15	507	0.03125	125	16236042 Loop: MAIN [{matmult.f90} {86,9}-{106,14}]
3.0	15	507	0.03125	125	16236042 MAIN [{matmult.f90} {39,15}] => Loop: MAIN [{matmult.f90} {86,9}-{106,14}]

... <rest deleted>



UNIVERSITY  
OF OREGON

# ParaProf visualization - callgraph



# ParaProf Visualization – Treetable view

TAU: ParaProf: Mean Statistics – /Users/khuck/data/surveyor/callpath					
	Name	Exclusive Time	Inclusive Time ▼	Calls	Child Calls
▼	MAIN	49,160.812	16,858,729.906	1	6.062
▼	Loop: MAIN [{matmult.f90} {86,9}-{106,14}]	507.284	16,236,042	1	4,000
	MPI_Recv()	15,464,563	15,464,563	2,000	0
	MPI_Send()	264,195	264,195	2,000	0
▼	Loop: MAIN [{matmult.f90} {117,9}-{128,14}]	2,368.968	16,124,446.032	1	194.548
▼	MULTIPLY_MATRICES	3,961.677	16,082,339.323	64.516	64.516
	Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]	16,078,377.645	16,078,377.645	64.516	0
	MPI_Recv()	27,289	27,289	65.516	0
	MPI_Send()	12,448.742	12,448.742	64.516	0
►	Loop: MAIN [{matmult.f90} {112,9}-{115,14}]	13,339.452	494,333.774	1	2,000
►	INITIALIZE	159	301,608	1	2
►	Loop: MAIN [{matmult.f90} {71,9}-{74,14}]	15,717	193,002	1	2,000
	MPI_Finalize()	117,369.062	117,369.062	1	0
	MPI_Init()	69,516.344	69,516.344	1	0
►	Loop: MAIN [{matmult.f90} {77,9}-{84,14}]	5,495	9,270	1	31
	MPI_Comm_rank()	70.125	70.125	1	0
	MPI_Comm_size()	48.062	48.062	1	0

# Adding multiple hardware counters

---

- Change the makefile:
  - `setenv TAU_MAKEFILE $TAU/Makefile.tau-multiplecounters-mpi-papi-pdt`
  - `export TAU_MAKEFILE=$TAU/Makefile.tau-multiplecounters-mpi-papi-pdt`
- Rebuild
  - `make clean`
  - `make`
- Submit the job (passing environment variables)
  - `qsub -t 5 -n 32 --env TAU_THROTTLE=1:COUNTER1=GET_TIME_OF_DAY:COUNTER2=PAPI_NATIVE_PNE_BGP_PU0_FPU_MULT_1:COUNTER3=PAPI_NATIVE_PNE_BGP_PU0_FPU_ADD_SUB_1 ./matmult`

# TAU output has changed...

---

- Instead of profile.x.x.x files, we have directories of files
- One directory for each hardware counter requested
  - MULTI\_\_GET\_TIME\_OF\_DAY
  - MULTI\_\_PNE\_BGP\_PU0\_FPU\_ADD\_SUB\_1
  - MULTI\_\_PNE\_BGP\_PU0\_FPU\_MULT\_1

# ParaProf derived metrics

TAU: ParaProf Manager

- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - papi/surveyor/data/khuck/Users/
          - GET\_TIME\_OF\_DAY
          - PNE\_BGP\_PU0\_FPU\_ADD\_SUB\_1
          - PNE\_BGP\_PU0\_FPU\_MULT\_1
          - PNE\_BGP\_PU0\_FPU\_MULT\_1 + PNE\_BGP\_PU0\_FPU\_ADD\_SUB\_1**
          - PNE\_BGP\_PU0\_FPU\_MULT\_1 + PNE\_BGP\_PU0\_FPU\_ADD\_SUB\_1 / GET\_TIME\_OF\_DAY
  - derby (jdbc:derby:/Users/khuck/src/tau2/apple/lib/perfdmf)
  - incite (jdbc:derby:/Users/khuck/src/tau2/apple/lib/perfdmf.incite)
  - openuh (jdbc:postgresql://spaceghost.cs.uoregon.edu:5432/openuh)
  - peri\_gtc (jdbc:postgresql://spaceghost.cs.uoregon.edu:5432/peri\_gtc)
  - perimilc (jdbc:postgresql://spaceghost.cs.uoregon.edu:5432/peri\_milc)
  - test (jdbc:postgresql://spaceghost.cs.uoregon.edu:5432/perfdmf\_test)

MetricField	Value
Name	PNE_BGP_PU0...
Application ID	0
Experiment ID	0
Trial ID	0
Metric ID	3

Argument 1: 0:0:0:3 - PNE\_BGP\_PU0\_FPU\_MULT\_1 + PNE\_BGP\_PU0\_FPU\_ADD\_SUB\_1

Argument 2: 0:0:0:0 - GET\_TIME\_OF\_DAY

Divide

Apply operation



# Saving profiles to a PerfDMF database

---

- IF you have a DBMS handy, use it!
- Otherwise, we include Derby support
  - Embedded database for Java applications
- 2 ways to configure:
  - `perfdmf_configure`
  - ParaProf (File -> Database Configuration)
- Configuration automatically creates the database if it doesn't exist for Derby – does not for others
- To save profiles:
  - `perfdmf_loadtrial`
  - ParaProf (context-clicks on database hierarchy)

# perfdmf\_loadtrial

---

Usage: perfdmf\_loadtrial -a <appName> -x <expName> -n <name> [options] <files>

## Required Arguments:

- n, --name <text> Specify the name of the trial
- a, --applicationname <string> Specify associated application name for this trial
- x, --experimentname <string> Specify associated experiment name for this trial

## Optional Arguments:

- c, --config <name> Specify the name of the configuration to use
- g, --configFile <file> Specify the configuration file to use  
(overrides -c)
- f, --filetype <filetype> Specify type of performance data, options are:  
profiles (default), pprof, dynaprof, mpip,  
gprof, psrun, hpm, packed, cube, hpc, ompp,  
snap, perixml, gptl
- m, --metadata <filename> XML metadata for the trial

## perfdmf\_loadtrial example

---

- `perfdmf_loadtrial -c incite -a matmult -x "My Experiment" -n "First Trial" .`
  - Uses incite PerfDMF configuration
    - `$HOME/.ParaProf/perfdmf.cfg.incite`
  - Uses “matmult” as application name
  - Uses “My Experiment” as experiment name
  - Uses “First Trial” as trial name
  - Expects default file type – TAU profiles
  - Uses “.” (current working directory) as profile location

# PerfExplorer with loaded trials

TAU: PerfExplorer Client

Analysis Management Cluster Results Correlation Results Custom Charts

Performance Data

- jdbc:derby:/Users/khuck/src/tau2/applications/matmult/scalability
  - 32
  - 64
  - 96
  - 128
  - 160
  - 192
  - 224
  - 256
- Views

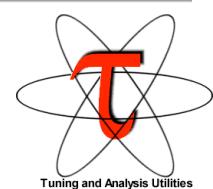
Field	Value
Name	32
Trial ID	1
DATE	ted
COLLECTORID	ted
NODE_COUNT	32
CONTEXTS_PER_NODE	1
THREADS_PER_CONTEXT	1
XML_METADATA	<?xml version="1.0" encoding="UTF-8"?> <tau:metadata xmlns:tau="...
XML_METADATA_GZ	ted

Element	Value
<tau:metadata>	
@ xmlns:tau	http://www.cs.uoregon.edu/research/tau
<tau:CommonProfileAttributes>	
@ BGP DDRSize (MB)	2048
@ BGP Node Mode	Coprocessor (17821024)
@ BGP Processor ID	0
@ BGP Size	(4,4,4)
@ BGP isTorus	(0,0,0)
@ BGP numNodesInPset	1
@ BGP numPsets	64
@ BGP psetNum	0
@ CPU Type	450 Blue Gene/P DD2
@ CWD	/gpfs/home/khuck/matmult/32
@ Executable	/sbin.rd/ioproxy
@ Hostname	ion-9
@ Local Time	2008-05-05T23:24:22+00:00
@ Memory Size	1773608 kB
@ Node Name	ion-9
@ OS Machine	BGP
@ OS Name	CNK
@ OS Release	2.6.19.2

O

UNIVERSITY  
OF OREGON

54



# Tracing with TAU

---

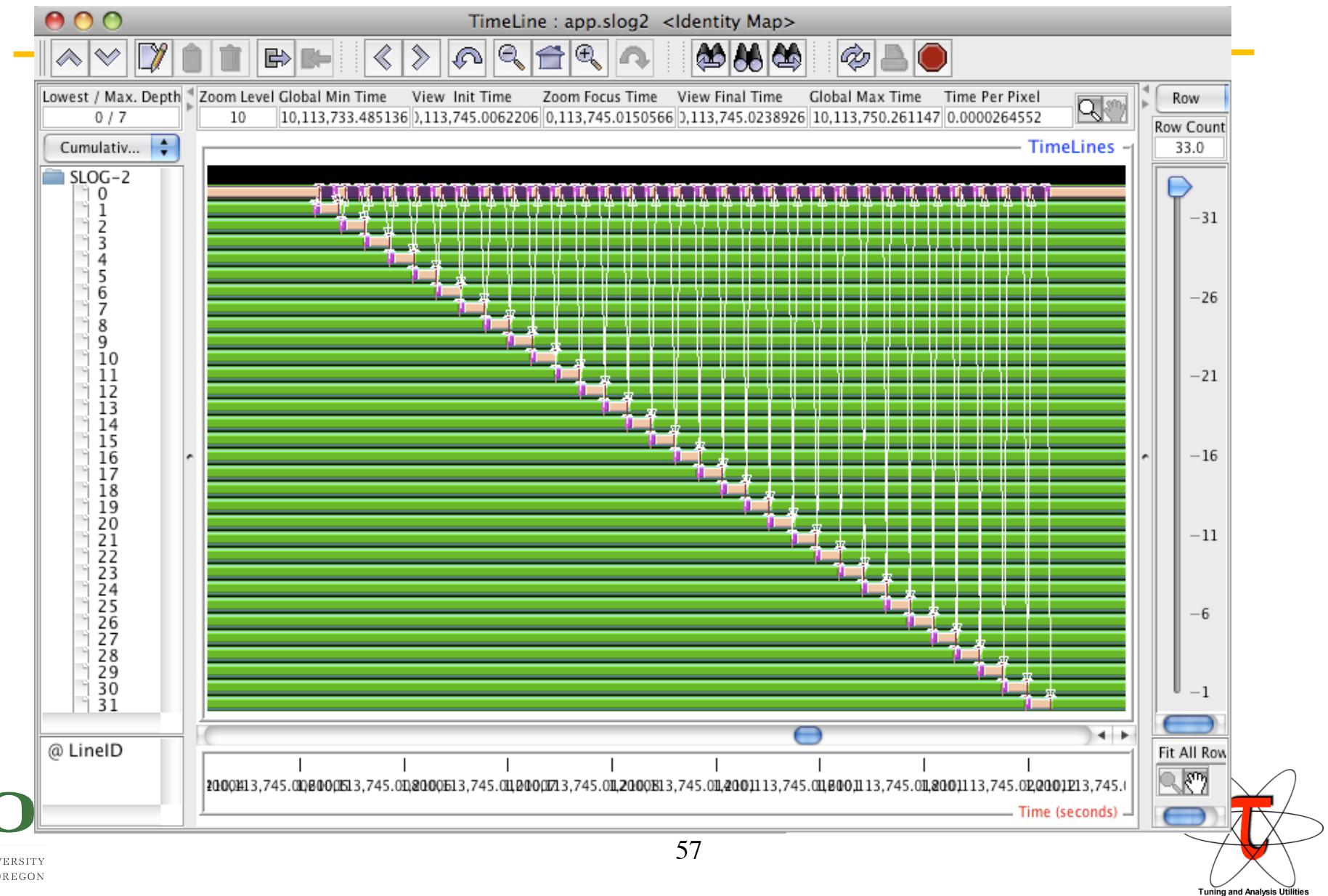
- Change the makefile:
  - `setenv TAU_MAKEFILE $TAU/Makefile.tau-mpi-pdt-trace`
  - `export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-trace`
- Rebuild
  - `make clean`
  - `make`
- Submit the job (passing environment variables)
  - `qsub -t 5 -n 32 ./matmult`

# TAU output

---

- 2 files per thread of execution
  - tautrace
  - events
- Need to merge the tracefiles:
  - tau\_treemerge.pl
- Need to convert the traces:
  - tau2slog2 tau.trc tau.edf -o app.slog
- To visualize, use jumpshot
  - jumpshot app.slog

# Jumpshot visualization



# Tutorial Summary

---

- Got example code
- Set up environment
- Built and ran experiments
- Visualized output with pprof
- Visualized output with ParaProf
- Saved output to PerfDMF
- Analysis with PerfExplorer
- Visualization with Jumpshot

# Acknowledgements

---

- National Science Foundation SDCI
- Department of Energy
- HPCMP DoD PET Program
- University of Tennessee
  - Shirley Moore
  - David Cronk
  - Karl Fuerlinger
  - Dan Terpstra
- University of Oregon
  - Allen D. Malony, S. Shende, A. Morris, W. Spear
- NCSA
- TU Dresden
  - Holger Brunst
  - Wolfgang Nagel
- Research Centre Juelich, Germany
  - Bernd Mohr
  - Felix Wolf
- Argonne National Lab

